Neural Networks, Vol. 2, pp. 459–473, 1989
Printed in the USA. All rights reserved.

0893-6080/89 $3.00 + .00

ORIGINAL CONTRIBUTION

Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network

TERENCE D. SANGER

Massachusetts Institute of Technology

(*Received* 31 *October* 1988; *revised and accepted* 26 *April* 1989)

Abstract—*A new approach to unsupervised learning in a single-layer linear feedforward neural network is discussed. An optimality principle is proposed which is based upon preserving maximal information in the output units. An algorithm for unsupervised learning based upon a Hebbian learning rule, which achieves the desired optimality is presented. The algorithm finds the eigenvectors of the input correlation matrix, and it is proven to converge with probability one. An implementation which can train neural networks using only local "synaptic" modification rules is described. It is shown that the algorithm is closely related to algorithms in statistics (Factor Analysis and Principal Components Analysis) and neural networks (Self-supervised Backpropagation, or the "encoder" problem). It thus provides an explanation of certain neural network behavior in terms of classical statistical techniques. Examples of the use of a linear network for solving image coding and texture segmentation problems are presented. Also, it is shown that the algorithm can be used to find "visual receptive fields" which are qualitatively similar to those found in primate retina and visual cortex.*

Keywords—Neural network, Unsupervised learning, Hebbian learning, Feedforward, Karhunen-Loeve Transform, Image coding, Texture, Cortical receptive fields.

INTRODUCTION

Research into the behavior of feedforward multilayer networks has increased dramatically since the discovery of the backpropagation learning algorithm (Rumelhart, Hinton, & Williams, 1986a, 1986b). Backpropagation allows us to train the weights in a feedforward network of arbitrary shape by following a gradient descent path in weight space, where the energy surface for the descent is usually defined by the mean squared difference between desired and actual outputs of the network. There have been many examples of successful use of backpropagation to perform nontrivial tasks (Hinton, 1987; Lippmann, 1987, for review).

Unfortunately, backpropagation has three major problems. The first is that the energy surface may have many local minima, so the algorithm is not guaranteed to converge to the optimal solution (Sontag, 1988; Sontag & Sussmann, 1988). The second problem is that it is often difficult to analyze the behavior of hidden units in such a network, since gradient descent imposes no particular structure on the solution. The third problem is that the backpropagation algorithm is often slow. The reason for the slow learning rests in the fact that the choice of weight for any unit depends on all the weights both above and below. Changing any single weight may require modification of all other weights in the network.

These considerations lead us to look for algorithms which can train each layer of the network "independently" of the layers above.[1] This suggests

Acknowledgements—The author would like to thank Tomaso Poggio, Rich Sutton, Steve Grossberg, Tom Breuel, Eero Simoncelli, Bror Saxberg, and Joel Wein for their comments and suggestions about the manuscript. The author was supported during the early part of this research by a National Science Foundation Graduate Fellowship, and later by a Medical Scientist Training Program grant. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124 and Army contract DACA76-85-C-0010. Additional support is provided by the Alfred P. Sloan Foundation, the Hughes Aircraft Corporation (S1-801534-2), and the NATO Scientific Affairs Division (0403/87).

Requests for reprints should be sent to Terence D. Sanger, MIT E25-534, Cambridge, MA 02139.

[1] For our purposes, a layer is defined to include a single layer of weights together with the following set of node functions. For example, a "two-layer" network has input, hidden, and output units separated by two sets of weights.

the use of unsupervised learning algorithms (e.g., Carpenter & Grossberg, 1988; Grossberg, 1976; Hinton, 1987; Kohonen, 1982). To train a multilayer network, we allow the training for each layer to depend only upon the outputs of the layer below. The entire multilayer network is then trained bottom-up, one layer at a time. This approach has been used by Linsker (1986) and Silverman and Noetzel (1988), for example. Optimal training for any layer will be defined in terms of the statistical properties of the outputs of that layer. These properties will depend upon the statistics of the outputs of the previous layer, and the weights between the two layers. Through bottom-up training, we can use an algorithm for training a single layer to successively train all the layers of a multilayer network.

In order to design each layer we need an "optimality principle." For this paper, we will assume that the network is linear, and that the structure is such that each layer has fewer outputs than inputs. Under these circumstances, we would like any given layer to preserve as much information as possible subject to the smaller number of outputs. Maximization of output information was first suggested by Linsker (1988) as a principle for the design of neural networks. For our purposes, we define an optimally trained layer as one which allows linear reconstruction of the inputs to that layer with minimal mean squared error. This is not equivalent to maximization of the Shannon information rate which Linsker proposes, but it is motivated by the same considerations.

With this optimality principle, we can find the optimal solution in closed form. The solution is given by the network whose weight vectors span the space defined by the first few eigenvectors of the autocorrelation matrix of the input. If the weights are the eigenvectors themselves, then the outputs will be uncorrelated and their variance will be maximized (subject to constraints on the weight magnitudes).

In this paper we propose a neural network training algorithm which converges to exactly this solution. We prove that the network converges from any initially random set of weights to find the eigenvectors of the input autocorrelation in eigenvalue order. The algorithm is called "The Generalized Hebbian Algorithm" (GHA), since it is based on well-known Hebbian algorithms for neural adaptation. We describe why such algorithms are related to eigenvector decomposition, and how the Generalized Hebbian Algorithm operates. We show how the algorithm can be implemented in a network with only local operations, and compare its behavior with that of other well-known algorithms in the literature. Finally, we present three examples of the application of single-layer networks trained with GHA to simple tasks in image coding, texture analysis, and feature extraction.

This paper shows the usefulness of eigenvector decomposition applied to neural networks and demonstrates the power of networks trained with the Generalized Hebbian Algorithm. While the properties of eigenvectors and the Karhunen-Loève Transform are well-known in many fields, the extensive applicability to neural networks has not yet been fully appreciated. In the following discussion and examples, we hope to provide some insight into the utility and practicality of this important technique.

1. GENERALIZED HEBBIAN ALGORITHM

We have developed an algorithm to train neural networks to find the eigenvectors of the autocorrelation matrix of the input distribution, given only samples from that distribution (Sanger, 1988). Each output of a trained network represents the reponse to one eigenvector, and the outputs are ordered by decreasing eigenvalue. A network trained in this way will allow linear reconstruction of the original input with minimal mean-squared error (see section 3).

Let the inputs to a single-layer network be an N-dimensional column vector x, the weights an $M \times N$ matrix C, and the outputs an M-dimensional column vector $y = Cx$ with $M < N$. Assume values of x are generated by a stationary white random vector stochastic process with correlation matrix $Q = E[xx^T]$. Therefore, x and y are both time-varying, and C will be time-varying as a result of adaptation through the training algorithm. (For readability, we will often not write the time argument explicitly.)

The "Generalized Hebbian Algorithm" (GHA) is given by:

$$c_{ij}(t+1) = c_{ij}(t) + \gamma(t)\left(y_i(t)x_j(t) - y_i(t)\sum_{k \le i} c_{kj}(t)y_k(t)\right). \quad (1)$$

We can write this in matrix form as:

$$\Delta C(t) = \gamma(t)\left(y(t)x^T(t) - \mathrm{LT}[y(t)y^T(t)]C(t)\right) \quad (2)$$

where LT[·] sets all elements above the diagonal of its matrix argument to zero, thereby making it Lower Triangular. Let $\gamma(t)$ be such that $\lim_{t\to\infty} \gamma(t) = 0$ and $\sum_{t=0}^{t=\infty} \gamma(t) = \infty$. Under these conditions, we can prove the following (see section 2 for proof):

Theorem 1: *If C is assigned random weights at time zero, then with probability* 1, *eqn* (2) *will converge, and C will approach the matrix whose rows are the first M eigenvectors of the input correlation matrix Q, ordered by decreasing eigenvalue.*

The significance of the theorem is that we now have a procedure which is guaranteed to find the eigenvectors which we seek. We do not need to compute the correlation matrix Q in advance, since the eigenvectors are derived directly from the data. This is an important feature, particularly if the number of inputs is large so that computation and manipulation of Q are not feasible. For instance, if a network has 4000 inputs, then $Q = E[xx^T]$ has 16 million elements, and it may be hard to find the eigenvectors by traditional methods. However, GHA requires the computation only of the outer products yx^T and yy^T, so that if the number of outputs is small the computational and storage requirements can be correspondingly decreased. If there are 16 outputs, for example, yx^T will have only 64,000 elements, and yy^T will have only 256. The Generalized Hebbian Algorithm takes advantage of this network structure. If the number of outputs of the network is close to the number of inputs, then the algorithm may not have a distinct advantage over other methods. But when the number of inputs is large and the number of required outputs is small, GHA provides a practical and useful procedure for finding eigenvectors.

In the field of signal processing, an almost equivalent algorithm was proposed in Owsley (1978), Oja (1983), and Oja and Karhunen (1985), although convergence to the matrix of eigenvectors was never proven. To our knowledge, the proof given here is the first convergence proof for this algorithm.

There exist several different but closely related algorithms for finding multiple eigenvectors which have been proven to converge (Brockett, 1989; Karhunen, 1984a, 1984b, 1985; Karhunen & Oja, 1982; Kuusela & Oja, 1982; Oja, 1983; Oja & Karhunen, 1980, 1985). Proofs of convergence for some of these algorithms, and an excellent summary of the different methods may be found in Oja (1983) and Karhunen (1984b). Recently, Brockett (1989) has proposed a related algorithm which is similar but which does not use the LT operator, and which he has proven converges to the eigenvectors in any desired order.

2. PROOF OF THEOREM 1

We must show that the algorithm

$$C(t+1) = C(t) + \gamma(t)\left(y(t)x^T(t) - \mathrm{LT}[y(t)y^T(t)]C(t)\right) \quad (3)$$

converges to the matrix T whose rows are the first M eigenvectors of $Q = E[xx^T]$ in descending eigenvalue order. C is an $M \times N$ matrix, $y(t) = C(t)x(t)$, LT[·] sets all entries of its matrix argument which are above the diagonal to zero, x is a white bounded vector stationary stochastic process with autocorrelation matrix Q, and $\{e_k\}$ is the set of eigenvectors of Q indexed by k in order of descending eigenvalue $\{\lambda_k\}$.

We will write

$$C(t+1) = C(t) + \gamma(t)h(C(t), x(t)) \quad (4)$$

where

$$h(C(t), x(t)) \doteq y(t)x^T(t) - \mathrm{LT}[y(t)y^T(t)]C(t).$$

We now apply theorem 1 of Ljung (1977) which states (in our notation):
If

1. $\gamma(t)$ is a sequence of positive real numbers such that $\gamma(t) \to 0$, $\Sigma_t\, \gamma(t)^p < \infty$ for some p, and $\Sigma_t\, \gamma(t) = \infty$;
2. $x(t)$ is bounded with probability 1;
3. $h(C, x)$ is continuously differentiable in C and x and its derivative is bounded in time;
4. $\bar{h}(C) \doteq \lim_{t\to\infty} E_x[h(C, x)]$ exists for each C;
5. S is a locally asymptotically stable (Lyapunov sense) set for the differential equation
$$\dot{C} = \bar{h}(C)$$
with domain of attraction $D\{S\}$, and
6. $C(t)$ enters some compact subset $A \subset D\{S\}$ infinitely often w.p.1;

Then with probability one,

$$\lim_{t\to\infty} C(t) \in S$$

where $C(t)$ is given by eqn (4). (A similar result is found in theorem 2.4.1 of Kushner & Clark, 1978.) This theorem allows us to study the evolution of a difference equation in terms of a related differential equation. This will simplify the discussion considerably. Note that Ljung's theorem does not apply directly to the convergence of matrices, but rather was written to apply to vectors. However, we can write the elements of the matrix C as a vector, and allow the nonlinear function h to perform the shape change from vector to matrix, and the theorem will then be directly applicable.

To satisfy the requirements of the theorem, we let $\gamma(t) = 1/t$ and, $\bar{h}(C) = \lim_{t\to\infty} E[h(C, x)] = CQ - \mathrm{LT}[CQC^T]C$. We now show that the domain of attraction $D\{S\}$ includes all matrices with bounded entries. We will "hard-limit" the entries of C so that their magnitudes remain below a certain threshold a, and thus within a compact region of R^{NM}. We will thereby show that C converges to the matrix of eigenvectors T for any choice of initial matrix $C(0)$.

We seek stable points of the differential equation

$$\dot{C} = CQ - \mathrm{LT}[CQC^T]C. \quad (5)$$

Fixed points exist for all C whose rows are permutations of any set of eigenvectors of Q. We will show that the domain of attraction of the solution given by $C = T$ whose rows are the first M eigenvectors in descending eigenvalue order, includes all matrices C with bounded entry magnitudes.

Oja showed that the first row of C will converge to the principal eigenvector with probability 1. (This is not the only fixed point, but it is the only asymptotically stable one.) We will use induction to show that if the first $i - 1$ rows converge to the first $i - 1$ eigenvectors, then the ith row will converge to the ith eigenvector. We assume that Q has N distinct strictly positive eigenvalues with corresponding orthonormal eigenvectors. (The case of repeated or zero eigenvalues is a straightforward generalization.)

The first row of the differential equation is given by:

$$\dot{c}_1^T = c_1^T Q - (c_1^T Q c_1) c_1^T$$

which is equivalent to eqn (7) of (Oja, 1982). Oja showed that this equation forces c_1 to converge with probability 1 to $\pm e_1$, the normalized principal eigenvector of Q.

At any time t, for $k < i$ write

$$c_k(t) = e_k + \varepsilon_k(t) f_k(t)$$

where e_k is the kth eigenvector (or its negative), f_k is a time-varying unit-length vector, and ε_k is a scalar. We assume for the induction step that for $k < i$, $\varepsilon_k(t) \rightarrow 0$ as $t \rightarrow \infty$. We must show that $c_i(t) \rightarrow e_i$ as $t \rightarrow \infty$.

Each row of eqn (5) can be written

$$\dot{c}_i = Qc_i - \sum_{k \le i} (c_i^T Q c_k) c_k.$$

Substituting $c_k = e_k + \varepsilon_k f_k$ gives

$$\dot{c}_i = Qc_i - (c_i^T Q c_i) c_i - \sum_{k<i} (c_i^T Q e_k) e_k - O(\varepsilon) + O(\varepsilon^2) \quad (6)$$

where $Qe_k = \lambda_k e_k$, and ε indicates a term converging to zero at least as fast as the slowest row for $k < i$. Expanding c_i in terms of the entire orthonormal set of eigenvectors gives

$$c_i = \sum_{k=0}^{N} \alpha_k e_k$$

where $\alpha_k = c_i^T e_k$. Inserting this expression gives the following equalities:

$$\begin{aligned}
(c_i^T Q c_i) c_i &= c_i \sum_{l=0}^{N} \lambda_l \alpha_l^2 \\
&= \left(\sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \sum_{k=0}^{N} \alpha_k e_k \\
Qc_i &= \sum_{k=0}^{N} \lambda_k \alpha_k e_k \\
Qc_i - \sum_{k<i} \lambda_k (c_i^T e_k) e_k &= \sum_{k=0}^{N} \lambda_k \alpha_k e_k - \sum_{k<i} \lambda_k \alpha_k e_k \\
&= \sum_{k \ge i} \lambda_k \alpha_k e_k.
\end{aligned}$$

We now assume t is large so we can ignore terms of order ε, and we write

$$\begin{aligned}
\dot{c}_i &= \sum_{k=0}^{N} \dot{\alpha}_k e_k \\
&= \sum_{k \ge i} \lambda_k \alpha_k e_k - \left(\sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \sum_{k=0}^{N} \alpha_k e_k \\
&= \sum_{k<i} \left(\sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \alpha_k e_k + \sum_{k \ge i} \left(\lambda_k - \sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \alpha_k e_k.
\end{aligned}$$

If we multiply each side by e_k^T for each k, the orthonormality of $\{e_k\}$ implies

$$\dot{\alpha}_k = \begin{cases} -\alpha_k \sum_{l=0}^{N} \lambda_l \alpha_l^2 & \text{if } k < i \\ \alpha_k (\lambda_k - \sum_{l=0}^{N} \lambda_l \alpha_l^2) & \text{if } k \ge i \end{cases} \quad (7)$$

We can use eqn (7) to study the recursion relation on the α_ks. We examine three cases: $k < i$, $k > i$, and $k = i$.

Since Q is positive definite, all the eigenvalues λ_k are positive, so the term $\gamma \sum_{k=0}^{N} \lambda_k \alpha_k^2$ is always greater than zero. Therefore, for $k < i$ we have

$$\dot{\alpha}_k = -\eta \alpha_k$$

where η is strictly positive. This expression will converge to zero for any initial value of α_k.

When $k > i$, define $\theta_k = \alpha_k / \alpha_i$. (Assume that $\alpha_i \ne 0$, which is true with probability one for randomly chosen initial weights $C(0)$.) We have

$$\begin{aligned}
\dot{\theta}_k &= (1/\alpha_i)(\dot{\alpha}_k - \theta_k \dot{\alpha}_i) \\
&= (1/\alpha_i) \left[\alpha_k \left(\lambda_k - \sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \right. \\
&\qquad \left. - \theta_k \alpha_i \left(\lambda_i - \sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \right] \\
&= \theta_k (\lambda_k - \lambda_i).
\end{aligned}$$

Since the eigenvalues are numbered in decreasing order, λ_i is the largest eigenvalue in $\{\lambda_i, \ldots, \lambda_N\}$, $\lambda_i > \lambda_k$ for all $k > i$, and we see that $\theta_k \rightarrow 0$ for $k > i$.

Next, we examine the behavior of α_i (case $k = i$). This is described by

$$\begin{aligned}
\dot{\alpha}_i &= \alpha_i \left(\lambda_i - \sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \\
&= \alpha_i \left(\lambda_i - \lambda_i \alpha_i^2 - \sum_{l \ne i} \lambda_l \alpha_l^2 \right).
\end{aligned}$$

But we know that $\alpha_k \rightarrow 0$ for $k < i$. We therefore drop terms in α_k for $k < i$, which gives

$$\begin{aligned}
\dot{\alpha}_i &= \alpha_i \left(\lambda_i - \lambda_i \alpha_i^2 - \sum_{l>i} \lambda_l \alpha_l^2 \right) \\
&= \alpha_i \left(\lambda_i - \lambda_i \alpha_i^2 - \alpha_i^2 \sum_{l>i} \lambda_l \theta_l^2 \right).
\end{aligned}$$

But $\theta_k \rightarrow 0$ for $k > i$, so the last term above approaches zero, and we therefore drop it as well, giving

$$\dot{\alpha}_i = \lambda_i(\alpha_i - \alpha_i^3).$$

To show that this converges, note that

$$V = (\alpha_i^2 - 1)^2$$

is a Lyapunov function, since

$$\dot{V} = 4\lambda_i(\alpha_i^3 - \alpha_i)(\alpha_i - \alpha_i^3) < 0$$

and V has a minimum at $\alpha_i = \pm 1$. Therefore, $\alpha_i \rightarrow \pm 1$ as $t \rightarrow \infty$. Since $\theta_k \rightarrow 0$ for $k > i$, we also have $\alpha_k \rightarrow 0$ for $k > i$. Thus for large t, the only significant α is α_i, so c_i will converge to $\pm e_i$, as desired. This completes the induction step and proves that the differential eqn (5) converges to the matrix T of eigenvectors of Q. The domain of attraction of T includes all matrices with bounded weights.

We must now show that there exists a compact subset A of the set of all matrices such that $C(t) \in A$ infinitely often with probability 1. Define the norm of C by $\|C(t)\| \doteq \max_{i,j} |c_{ij}(t)|$. Let A be the compact subset of R^{NM} given by the set of matrices with norm less than or equal to some constant a. It can be shown that for a sufficiently large, if $\|C(t - 1)\| > a$ then $\|C(t)\| < \|C(t - 1)\|$ w.p.1. Thus C will eventually be within A, and it will remain inside A (infinitely often) with probability 1, as t becomes large. Since the domain of attraction includes all matrices with bounded norm, $A \in D\{S\}$ and condition (6) of Ljung's theorem is satisfied.

We have now satisfied all the conditions of Ljung's (1977) theorem. This proves that, under the assumptions given above, eqn (3) will cause C to converge with probability 1 to the matrix T whose rows are the first M eigenvectors in descending eigenvalue order.

Note that theorem 2.3.1 of Kushner and Clark (1978) and the assumptions given above imply that the averaged form of the algorithm, given by

$$C(t + 1) = C(t) + C(t)Q - \mathrm{LT}[C(t)QC^T(t)]C(t) \quad (8)$$

will also cause C to converge with probability 1 to the matrix of eigenvectors.

3. OPTIMALITY

Linsker (1988) has suggested that maximization of the output information may be a fundamental principle for the organization of biological neural networks. Inspired by this idea, we have defined "optimal" unsupervised learning as learning which allows us to linearly reconstruct the inputs to a layer from its outputs with minimal error. To make this more precise, we will make some specific assumptions about the structure of the neural networks we consider, and how we want them to behave.

We now consider only single-layer linear feedforward networks whose operation is described by the equation $y = Cx$ where x represents the input data (a white vector stochastic process), and the network computes the outputs y. We assume that the number of inputs and outputs has been specified, and that there are fewer outputs than inputs. The only aspect of the network which is subject to training is the matrix C which should be adapted according to some goal for network processing. We choose this goal to be that the outputs allow subsequent linear processing to reconstruct the input x with minimal mean-squared error. The assumption that there are fewer outputs than inputs requires that there will be some error, but a proper choice of the matrix C can reduce it. The assumption that subsequent processing must be linear means that this criterion is not equivalent to maximizing the Shannon information in the outputs, except in certain special cases (Linsker, 1988).

We know the optimal choice of weights which minimizes the linear reconstruction error from standard results in linear algebra. Let x be a randomly chosen vector of N inputs, C an $M \times N$ matrix of weights, and y the vector of $M < N$ outputs such that $y = Cx$. Assume that x is zero-mean with correlation matrix Q. Then the linear least squares estimate (LLSE) of x given y is

$$\hat{x} = QC^T(CQC^T)^{-1}y$$

and the mean squared error $E[(x - \hat{x})^2]$ is minimized when the rows of C span the first M eigenvectors of Q (Bourlard & Kamp, 1988; Fukunaga, 1972; Golub & Van Loan, 1983; Kazakos, 1983).

If the rows of C actually are the first M eigenvectors, then $CC^T = I$, and $Q = C^T\Lambda C$ where Λ is the diagonal matrix of eigenvalues of Q in descending order. This defines the "eigenvector decomposition" of Q, and we say that $y = Cx$ is the "Karhunen-Loève Transform" (KLT) of the input. This is what the Generalized Hebbian Algorithm accomplishes. The correlation matrix of the outputs is $CQC^T = \Lambda$ which is diagonal, so we see that the outputs are uncorrelated and have variance equal to the eigenvalues. (See Watanabe, 1965 for a theoretical description of the KLT and some remarks on its applications.)

Very often in designing a neural network we do not know how wide to make a given layer of the network. If a layer has been trained to compute the KLT, then all the outputs are uncorrelated, and their variance (eigenvalue) is a measure of the "usefulness" of that particular output. The entropy term

$$-\sum_{i=1}^{M} E[y_i^2]\log E[y_i^2]$$

is minimized for all values of M (Watanabe, 1965), so that information is concentrated in the first few outputs. The variance of the outputs decreases quickly with i, and we can select only the first few outputs whose variance is greater than some chosen threshold. Once we find an output whose variance is below the threshold, we can stop computing additional terms since all later outputs will have variances which are even lower and may decrease our approximation accuracy (Brailovsky, 1983a, 1983b, 1985). We can therefore choose the width of a layer based on the actual information conveyed by the outputs of that layer. For information transmission, we can quantize the different outputs with different numbers of bits and thereby gain significant compression factors (see section 7.1 for an example).

The KLT can give some understanding of what a network has learned. Since the high-variance outputs represent significant components of the input, it seems reasonable to believe that these outputs respond to "features" in the input space which convey useful information about the data. The algorithm will find vectors which "explain the variance" in the input. As mentioned above, these vectors are useful for reconstructing the original data from fewer outputs. But they can also be used to analyze the data and separate it into uncorrelated components which may represent underlying physical processes in the environment (see sections 7.2 and 7.3 for examples).

4. HEBBIAN ALGORITHMS

The Generalized Hebbian Algorithm is closely related to classical Hebbian Learning algorithms. Hebbian learning rules modify the connection between two units by an amount proportional to the product of the activation of those units (Hebb, 1949). In our notation, if x is the activation of the input nodes and C is the weight matrix, then $y = Cx$ is the activation at the outputs. Hebbian algorithms modify C using

$$C(t + 1) = C(t) + \gamma\left(y(t)x(t)^T\right) \qquad (9)$$

where γ determines the rate of change of the weights.

Oja has shown that if we maintain the diagonal elements of CC^T equal to 1 (so that the norm of each row is 1), then a Hebbian learning rule will cause the rows of C to converge to the principal eigenvector of Q (Oja, 1982, 1983; Oja & Karhunen, 1980). He proposed a network learning algorithm (Oja, 1982):

$$\Delta c_i = \gamma(y_i x - y_i^2 c_i) \qquad (10)$$

where c_i^T is a row of C, and $y_i = c_i^T x$ (Oja, 1982). (In the following, we will often refer to (10) as "the Oja learning rule.") Oja claims that (10) can be approximated under certain conditions on x and γ by a differential equation (in our notation)

$$\dot{c}_i(t) = Qc_i(t) - (c_i(t)^T Q c_i(t))c_i(t) \qquad (11)$$

where $Q = E[xx^T]$. He then proves that for an arbitrary choice of initial weights, c_i will converge to the principal eigenvector e_1 (or its negative) so long as $c_i(0)^T e_1 \neq 0$ at time zero (Oja, 1982, 1983; Oja & Karhunen, 1980).

Hebbian learning rules tend to maximize the variance of the output units. The response to the principal component $y_1 = e_1^T x$ has larger variance than the response to any other unit-length vector. The variance of the output is $E[(e_1^T x)^2] = e_1^T Q e_1$. To see why the Oja algorithm maximizes the variance, we can define an energy function by

$$\mathcal{E} = -\frac{1}{2} c_i^T Q c_i.$$

This is just the variance of the output y_i, so if $\mathcal{E}$ is minimized subject to $c_i^T c_i = 1$, then the output $y_i = c_i^T x$ has maximum variance subject to this constraint. If we perform gradient descent on $\mathcal{E}$ for each element c_{ij} of c_i we obtain

$$\begin{aligned} \frac{\partial c_{ij}}{\partial t} &= -\frac{\partial \mathcal{E}}{\partial c_{ij}} \\ &= [Qc_i]_j \end{aligned}$$

where $[Qc_i]_j$ is the jth element of the vector Qc_i. We can then write

$$\frac{\partial c_i}{\partial t} = Qc_i$$

which we see is the first term of (11). This term is therefore performing gradient descent on an energy function which will maximize the output variance. The second term tends to keep $c_i^T c_i$ close to 1. The combination of the two terms has fixed points at all eigenvectors, but the only asymptotically stable solution is e_1, which maximizes the variance (Oja, 1982).

The Oja algorithm only finds the first eigenvector, so if we are given a network with more than one output, we need an extension of this algorithm which will allow us to find the other eigenvectors. The Generalized Hebbian Algorithm was designed to do this by combining the Oja learning rule (10) and a Gram-Schmidt orthogonalization process. We can write the Gram-Schmidt process in matrix form as follows:

$$\Delta C(t) = -\text{lower}[C(t)C^T(t)]C(t) \qquad (12)$$

where C is an $M \times N$ matrix whose rows are to be orthogonalized, t indexes discrete steps, and it is assumed that the rows of C are kept normalized to length 1 by some other process. The lower[·] operation sets all elements on or above the diagonal of its matrix argument to zero. Equation (12) will orthogonalize C in $M - 1$ steps if the row norms are maintained equal to 1.

The Generalized Hebbian Algorithm (2) is a combination of (12) and the matrix version of the Oja algorithm, given by

$$\dot{C}(t) = C(t)Q - \mathrm{diag}(C(t)QC(t)^T)C(t)$$

which applies the Oja algorithm to each row of C independently and thus causes all rows to converge to the principal eigenvector.

5. LOCAL IMPLEMENTATION

To help understand the operation of the Generalized Hebbian Algorithm, we rewrite (1) as

$$\Delta c_{ij}(t) = \gamma(t)y_i(t)\left(x_j(t) - \sum_{k<i} c_{kj}(t)y_k(t)\right) - \gamma(t)y_i^2(t)c_{ij}(t). \quad (13)$$

In this form, we see that the algorithm is equivalent to performing the Oja (1982) learning rule (10) using a modified version of the input given by

$$x_j'(t) = x_j(t) - \sum_{k<i} c_{kj}(t)y_k(t) \quad (14)$$

or, in matrix form

$$x'(t) = x(t) - \sum_{k<i} c_k(t)y_k(t)$$

where c_k is the kth row of the matrix C. The modified input which trains output i is formed by subtracting the components $c_k(t)$ which contributed to the previous outputs $y_k(t)$ for $k < i$. If the first $i - 1$ outputs respond to the first $i - 1$ eigenvectors, then the ith output "sees" an input from which those eigenvectors have been removed. The principal component of the modified input is now the ith eigenvector of the original input. When the Oja algorithm is applied to the modified input, it causes the ith output to learn the ith eigenvector, as desired. (This method of finding successive eigenvectors is similar to a technique known as "Hotellings's Deflation" (Kreyszig, 1988).)

Equation (13) shows how we can implement the algorithm using only local operations. This ability is important for training neural networks using parallel hardware. We can compute the outputs in order for each training presentation, and subtract the components $c_k(t)y_k(t)$ progressively from the input as we go. This corresponds to "using up" some of the input "energy" as we train each of the outputs. A local synapse-like structure which can perform this operation is shown in Figure 1. A given input x_1 is connected to several outputs. As each output y_i is activated, it inhibits the input unit x_1 by an amount proportional to the output activation. (The weights for forward propagation and reciprocal inhibition must be maintained equal and be modified together.) If the outputs learn in sequence, then each subsequent output sees an attenuated input. This leads to training according to (13) which is performed using only local "synaptic" operations. The fact that such a local implementation exists for this algorithm distinguishes it from other algorithms for computing the Karhunen-Loève transform and contributes to its importance for training neural networks.

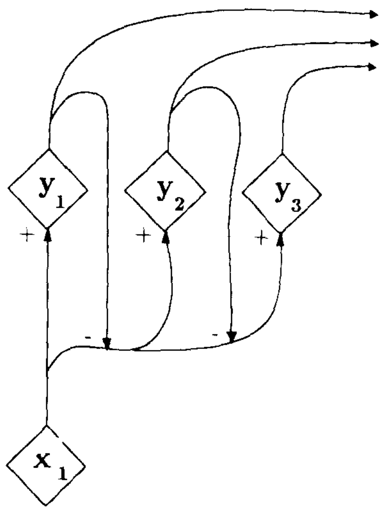

FIGURE 1. Network implementation of the Generalized Hebbian Algorithm.

Although we do not have any quantitative results concerning the rate of convergence to acceptable values of the matrix C, we can make some qualitative predictions. Training of a particular output is dependent on the training of the other outputs only through the modifications of the input given by (14). Therefore, we would expect all the outputs to train at the same rate independently of each other, so that training time for the entire network should be independent of the number of outputs. (Note that successive outputs do not "wait" for previous ones to converge, but rather learn simultaneously.) However, modification of the input by (14) effectively decreases the variance of the input used to train higher-numbered outputs. Decreasing the variance of the input is mathematically equivalent to decreasing γ, so we expect that the learning rate should decrease for the later outputs. In practice, training time (in terms of the number of input samples) is roughly proportional to the number of outputs.

6. RELATED ALGORITHMS

6.1. Eigenvector Decomposition

There are many fields which make use of eigenvector decomposition (the Karhunen-Loève Transform), and there are many algorithms for computing it. In statistics, the techniques of Factor Analysis and Principal Components Analysis (PCA) are commonly used tools which are closely related to the KLT. Most algorithms involve estimating the data correlation matrix Q and then finding the eigenvector decomposition using matrix techniques (see Golub and Van Loan (1983) or Kreyszig (1988) for review). These methods are computationally difficult if the number of inputs is large, since Q becomes unmanageable (see section 1). Algorithms exist which can find the eigenvectors given only samples of the input distribution, without the need to explicitly compute Q. Oja (1983) and others proposed an iterative method called "Stochastic Gradient Ascent" based on multiplication by an estimated correlation matrix followed by Gram-Schmidt orthonormalization. Another such algorithm involves finding the first eigenvector (principal component) using a method equivalent to the Oja algorithm (10), subtracting this component from the data sample, and then learning the next component (Kreyszig, 1988). This is equivalent to the procedure used here, except that here all the components converge at the same time. A summary of other related algorithms may be found in (Oja, 1983).

To our knowledge, the Generalized Hebbian Algorithm (GHA) is the first KLT algorithm for which a network implementation exists that can be used to train a neural network using only local operations. This may be the first application of such an algorithm to the field of neural networks. It must be remembered that in actual use with finite training times, GHA will only be able to approximate the eigenvectors and that errors in finding the first few eigenvectors will magnify the errors in finding subsequent eigenvectors. In other words, the algorithm has poor numerical accuracy for all but the first few eigenvectors. If greater accuracy is needed, or exhaustive computation of all the eigenvectors is desired, then classical matrix techniques may be more suitable. But if the input has high dimensionality, samples are readily available, and only the first few eigenvectors are needed, then GHA provides a considerably easier and faster computational alternative.

6.2. Winner-Take-All Networks

Winner-take-all networks which are based upon Hebbian learning rules may have some functional similarities to the algorithm presented here. In a winner-take-all network, usually only the output with largest value has its weights modified. This technique is used to ensure that different outputs learn different functions of the input (Barrow, 1987; Grossberg, 1976; Kohonen, 1982, 1988, among others). Although the convergence properties of such algorithms are often not well understood, we suspect that there may be a strong underlying relationship between the Generalized Hebbian Algorithm and winner-take-all algorithms. The learning rule is often not explicitly Hebbian, but rather modifies the weights of the winning unit to move them closer to the input which caused it to win. But for a unit to win, it must have a positive output, and therefore modifying the weights by the input is equivalent up to a scale factor to true Hebbian learning. We would therefore expect that any output would tend to converge to the principal eigenvector of the subset of the input data for which that output wins. Different outputs would converge to the principal eigenvectors of different subsets of the data. Note that this is not the same as converging to other eigenvectors, since different outputs will usually not be uncorrelated.

6.3. Self-Supervised Backpropagation

Several authors have experimented with the technique of Self-supervised Backpropagation (SSBP), also known as the "encoder" problem (e.g., Ballard, 1987; Cottrell, Munro, & Zipser, 1987; Hinton, 1987). This algorithm seems to have optimal data coding properties similar to those of the algorithm presented here. Its prevalence in the literature merits a detailed discussion of the relation to the Generalized Hebbian Algorithm (GHA).

In linear SSBP, a two-layer network is trained to perform the identity mapping, yet the number of hidden units is set to be fewer than the number of inputs. The hidden units must therefore discover an efficient encoding of the input data. Since efficient coding is also the goal for the Generalized Hebbian Algorithm, we would expect both algorithms to produce similar results. Bourlard and Kamp (1988) have shown that a set of vectors which span the Singular Value Decomposition (which is equivalent to the KLT as used here) gives the optimal set of hidden units of such a network. Baldi and Hornik (1989) proved that the backpropagation energy function has a unique minimum at this solution. There are many possible sets of weights, however, and SSBP will choose one based on the initial random choice of weights for the network. The GHA finds the unique set of weights which is both optimal and gives uncorrelated outputs. Therefore it is clear that in the linear case, SSBP and the GHA converge to almost equivalent solutions. We will now show that the equations describing the two algorithms are similar.

Define a three-layer linear network of N input units x, M hidden units y, and N output units $\hat{x}$. The weight matrices are W_1 and W_2, so that $y = W_1 x$ and $\hat{x} = W_2 y$. We train the network using backpropagation and error function $E[(x - \hat{x})^T(x - \hat{x})]$. Then, using the notation of Rumelhart et al. (1986a), we adapt the weights at each layer using $\Delta W_1 = \nu_1 \delta_1 x^T$ and $\Delta W_2 = \nu_2 \delta_2 y^T$ where $\delta_2 = x - \hat{x}$ and $\delta_1 =$

$W_2^T\delta_2$. We then have:

$$\begin{aligned}\Delta W_2 &= \nu_2(x - \hat{x})y^T\\ &= \nu_2(x - W_2y)y^T\\ &= \nu_2(xy^T - W_2yy^T)\\ \Delta W_2^T &= \nu_2(yx^T - yy^TW_2^T). \qquad (15)\end{aligned}$$

This is similar to (2), except that we do not force the matrix yy^T to be lower triangular, and W_2^T is not the matrix used to generate y from x. The following discussion will show that W_2^T and W_1 are related so that it is reasonable to assume (15) is almost equivalent to (2).

Assume that $\nu_2 \ll \nu_1$, so that W_2 converges much slower than W_1. We then have:

$$\begin{aligned}\Delta W_1 &= \nu_1W_2^T\delta_2x^T\\ &= \nu_1W_2^T(x - \hat{x})x^T\\ &= \nu_1W_2^T(x - W_2W_1x)x^T\\ &= \nu_1(W_2^Tx - W_2^TW_2W_1x)x^T\\ &= \nu_1(W_2^T - W_2^TW_2W_1)xx^T\\ E[\Delta W_1] &= \nu_1(W_2^T - W_2^TW_2W_1)Q.\end{aligned}$$

Since Q is positive definite, it can be shown that this equation will cause $W_2^TW_2W_1$ to approach W_2^T (for ν_1 decreasing to 0 as $1/t$). If we substitute $W_2^T \approx W_2^TW_2W_1$ into (15), we see that this is similar to (2) except for the lack of the LT[] operator and the inclusion of the positive definite scaling matrix $W_2^TW_2$.

Although the above discussion is not a rigorous proof, it gives us some insight into the reason for the close relationship between GHA and SSBP. The two algorithms are related both by the results they achieve and by their mechanisms of action. This means that our understanding of the convergence properties of GHA can be used to analyze certain backpropagation networks as well.

The major difference in function between GHA and SSBP is that GHA produces the first M eigenvectors themselves in eigenvalue order, while SSBP produces a linear combination of the first M eigenvectors. This difference can be important. Several authors have noted that SSBP tends to produce hidden units which have approximately equal variance (Baldi & Hornik, 1989; Cottrell et al., 1987). The variances do not descend by eigenvalue as they do for the KLT. The solution may not be unique, and although it spans the first few eigenvectors, the actual matrix which is learned cannot be predicted. In addition, the hidden units are not uncorrelated. It may be difficult to interpret the network as representing significant features of the environment, since the hidden units and their correlations may depend upon the initial randomly chosen weights as well as on the sequence of training examples.

These factors may reduce the usefulness of SSBP-trained hidden units for data coding applications. Because the hidden units all have approximately equal variance, bits must be allocated evenly among them, and noise cannot be eliminated by removing the units with lowest variance. If the network is designed with too many hidden units (in the sense of Brailovsky, 1983a, 1983b, 1985) then the additional error introduced is spread evenly throughout the units and cannot be easily detected or removed by looking at the signal to noise ratio of the individual units. Cottrell et al. (1987) point out that if channel errors affect certain units more than others, then it may be an advantage to distribute the information evenly so that high-variance channels are not corrupted excessively. If multiplicative noise is present in different amounts on different channels, then indeed this will be true. For additive noise, however, the KLT allows much easier reconstruction of the original "clean" signal, since the signal-to-noise ratio is maximized.

7. EXAMPLES

We now present examples of the use of single-layer linear networks which have been trained using the Generalized Hebbian Algorithm. The networks are used to solve problems in image coding, texture segmentation, and receptive field modeling.

7.1. Image Coding

Figure 2 shows an original image taken from part of an Eisenstadt photograph and digitized to form a 256 × 256 image with 256 greylevels. We use a single-

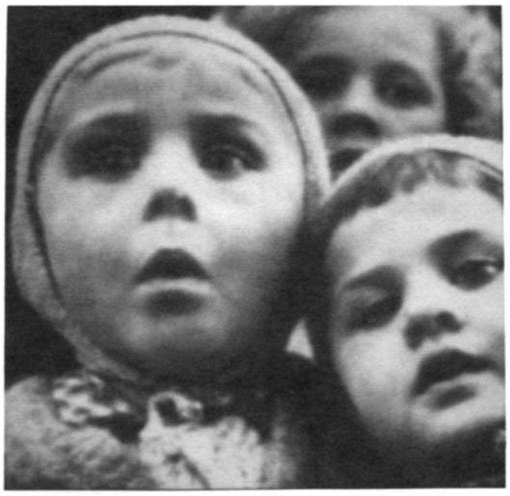

FIGURE 2. 256 × 256 pixel (8 bit) test image for coding.

layer linear neural network with 64 inputs and 8 outputs. 8 × 8 blocks of the image are used as training samples, with the image being scanned from left to right and top to bottom. The sample blocks do not overlap, and the image is scanned twice to allow time for the network to converge (giving 2048 samples). (Although in principle the rate term γ in the algorithm should decrease to zero as $1/t$, in all of the examples γ is held fixed at a value between 0.1 and 0.01 which is chosen empirically to provide good convergence, and which depends on the number of inputs and the average input variance.)

The weights which the network learns are represented as 8 × 8 masks shown in Figure 3. Each mask shows the set of weights associated with a single output. White indicates positive weights, black indicates negative, and grey indicates zero. In our notation, the masks are the rows c_i of the 8 × 64 weight matrix C after it has converged.

To code the image, each 8 × 8 block of the image is multiplied by each of the eight masks to generate eight coefficients for coding. The coefficients for the block starting at position n, m in the image I are thus given by

$$y_i^{nm} = \sum_{p=1}^{8} \sum_{q=1}^{8} c_{i,p+8q} I_{n+p,m+q}.$$

The coefficients $y_i^{n,m}$ are then uniformly quantized with a number of bits approximately proportional to the log of the variance of that coefficient over the image. This results in the first two masks being assigned five bits each, the third mask three bits, and the remaining masks two bits each. Therefore, each 8 × 8 block of pixels is coded using a total of 23 bits, so the data rate is 0.36 bits per pixel. To reconstruct the image from the quantized coefficients $\hat{y}_i^{n,m}$, each block of the image is re-formed by adding together all the masks weighted by their quantized coefficients:

$$\hat{I}_{n+p,m+q} = \sum_{i=1}^{8} c_{i,p+8q} \hat{y}_i^{n,m}.$$

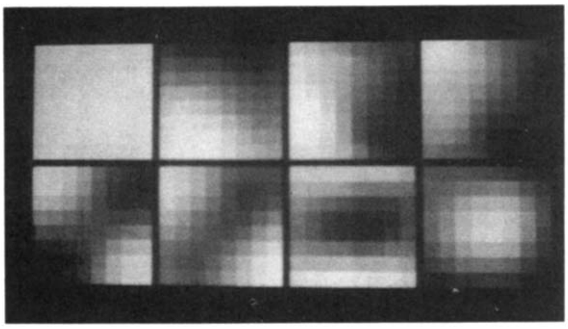

FIGURE 3. 8 × 8 masks learned by a network trained on Figure 2.

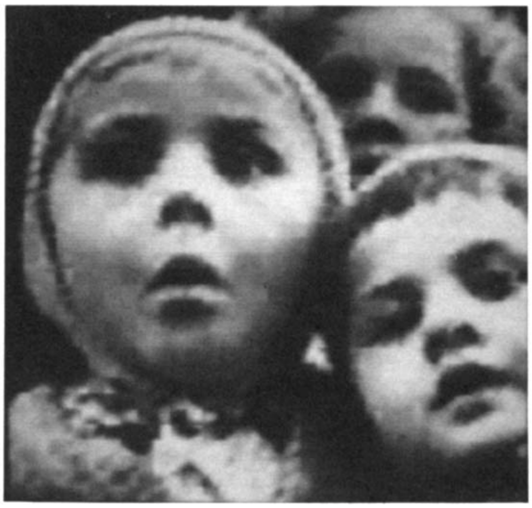

FIGURE 4. Image of Figure 2 coded at .36 bits per pixel.

The resulting image is shown in Figure 4. We calculate the normalized mean square error (NMSE) as in (Cottrell et al., 1987) to be the ratio of the error variance to the data variance

$$\text{NMSE} = \frac{E[(I_{n,m} - I_{n,m})^2]}{E[I_{n,m}^2]}$$

which is 0.043 for this image.

The network has learned a linear coding for the input data which approximates the optimal KLT. The masks are the "eigenvectors" of the input image, and they represent most of the variance in the 8 × 8 blocks which were used for training. Because the network outputs have high variance, they convey much of the input information, and we only need to use a few outputs (eight, in this case) to estimate the input data. This is the meaning of data coding; we have reduced 64 eight-bit pixels (512 bits total) to eight coefficients quantized with from two to five bits (23 bits total). The network chose masks which allow only 23 bits to represent most of the information in the original 512 bits.

We might now ask whether this same set of masks would be useful on a different image. Figure 5 is an image of a dog, and Figure 6 shows the image after it has been reconstructed from quantized coefficients derived from the set of masks in Figure 3. Note that the network was never trained on the image of Figure 5. In this case, the output of the first two masks was quantized using seven pixels each, the third with five pixels, the fourth with four pixels, and the remaining coefficients with three pixels each. This gives a total of 35 bits, or a bit rate of 0.55 bits per pixel. The NMSE is 0.023 here which is actually lower than the error for the image of Figure 1, due to the increased number of bits used for coding. The fact that the same set of masks can be used to code two different pictures is an example of "generalization" of the net-

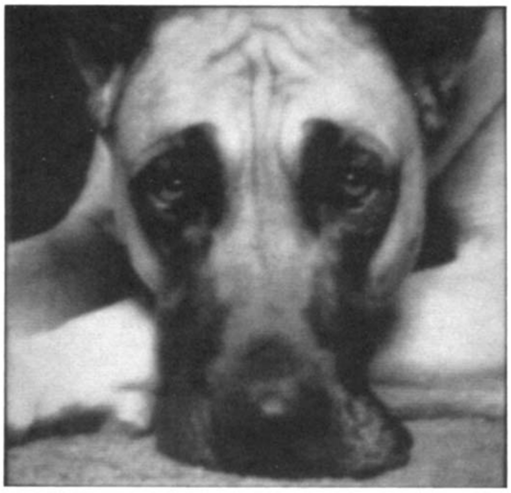

FIGURE 5. 256 × 256 pixel (8 bit) test image for coding.

work. Although the images are different, their statistics may be similar enough that their respective KLTs are similar. A network trained on either image will compute a set of masks which will be useful for the other. This generalization property is a direct consequence of the statistical similarity of the two images.

Filters similar to those given in Figure 3 have been used for image coding by many authors. The Discrete Cosine Transform masks are qualitatively similar (see Lim, in press for a description), and Daugman (1988) has performed image coding using two-dimensional Gabor filters which are also similar to the masks which the network learned.

Cottrell et al. (1987) used self-supervised backpropagation to perform image coding, with bit rates as low as 0.625 bits per pixel. They used a network with 64 inputs, 8 hidden units, and 64 outputs. Their training samples were, as here, 8 × 8 blocks of the image, and the network was trained to approximate the input data at the output units. As mentioned above, such a network will not actually find the KLT, but will tend to converge to outputs which represent linear combinations of the KLT vectors (Baldi & Hornik, 1989). Since the hidden units will all have approximately equal variance (Baldi & Hornik, 1989; Cottrell et al., 1987), it is not possible to quantize them with different numbers of bits, as it was for the KLT. This fact significantly reduces the maximum compression rate which can be achieved. It should also be noted that Cottrell et al. (1987) trained their network for 150,000 iterations, while the network which learned the masks of Figure 3 was trained for 2048 iterations.

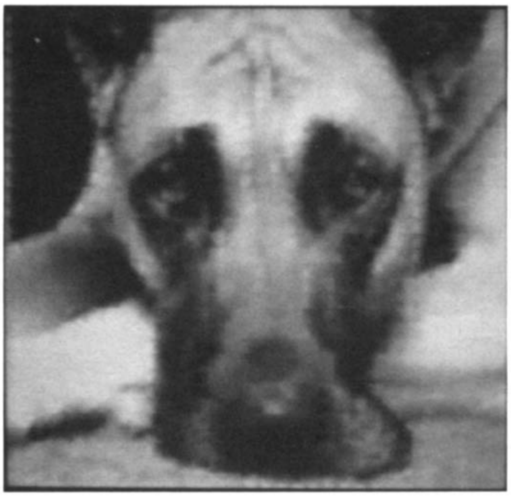

FIGURE 6. Image of Figure 5 coded at .55 bits per pixel using the same masks as in Figure 3.

7.2. Texture Segmentation

Since the Generalized Hebbian Algorithm finds the eigenvectors of the input distribution, we would expect that the outputs of the network will respond to "significant" features of the environment. The outputs will have high variance, and can be used for separating the input into different classes. To illustrate these ideas, we demonstrate the use of such a network to perform a simple texture segmentation task.

Figure 7 shows a 128 × 128 image formed from two different textures consisting of randomly placed horizontal and vertical line segments. (The image has been filtered with a narrow Gaussian with standard deviation of one-half pixel, since this was found to improve the convergence rate of the network.) We construct a single-layer linear network with 64 inputs and 4 outputs. The network is trained on 1000 overlapped 8 × 8 blocks of the image. However, before presenting any block to the network, it is multiplied by a Gaussian window with standard deviation of two pixels. This is done so that the network will become sensitive to the central region of any block. The weights which are learned are shown in Figure 8a represented as 8 × 8 masks. The first mask is a low-pass filter, the second and third are horizontal

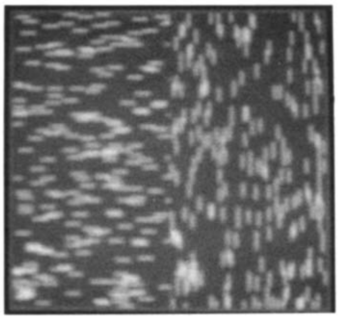

FIGURE 7. 128 × 128 test image for texture segmentation.

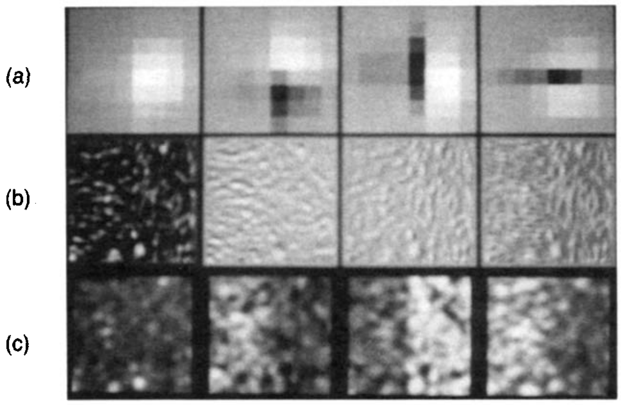

FIGURE 8. (a) masks learned by a network trained on Figure 7. (b) Convolution of masks in (a) with Figure 7. (c) Estimate of local variance of (b).

and vertical "edge-detectors," and the fourth is a horizontal "bar-detector."

We next convolve each of the masks with the original image in Figure 7 to show which regions of the image give high response for each mask. The convolution results are shown in Figure 8b. To estimate the local variance, we full-wave rectify the images of Figure 8b, and low-pass filter the results with a Gaussian of standard deviation 4 pixels. This gives a measure of local "energy" or variance, and the result is shown in Figure 8c for each mask. Here we can clearly see that certain masks respond preferentially to one or the other of the two textures in the image. The first mask is a low-pass filter, which achieves equal and maximal variance everywhere in the image. The next three masks have larger variance when responding to a particular texture (lines of one or the other orientation). Note that the texture preference of a mask is not immediately apparent from the convolution result in Figure 8b. We must approximate the local variance as in Figure 8c in order to distinguish the texture regions.

An equivalent technique was developed by Turner (1986), although he used a fixed set of masks which were not learned. His masks were Gabor filters, and he computed the squared amplitude of the response to a given filter at different points in the image. This operation is almost equivalent to taking the absolute value and low-pass filtering as we have done here. Both techniques find an estimate of the local amplitude (variance) of the response to a filter. Gabor filters were also used in (Daugman, 1988) to perform texture segmentation, although the local variance or response amplitude was not explicitly computed. Similarly, Voorhees (1987) and Voorhees and Poggio (1988) used $\nabla^2 G$ filtering to discriminate texture regions, while Bergen and Adelson (1988) used a center-surround with a rectification nonlinearity.

7.3. Receptive Fields

There have been many attempts to provide mechanisms which explain the development of the receptive fields of retinal and visual cortical cells (Barrow, 1987; Bienenstock, Cooper, & Munro, 1982; Kammen & Yuille, 1988; Linsker, 1986; Yuille, Kammen, & Cohen, in press, among others). We here present yet another attempt. An important difference between our method and most previous ones is that we not only provide a model for the development of receptive fields, but we show that this model implies that these receptive fields have optimality properties associated with the KLT.

The training examples are similar to those used in (Linsker, 1986). Each sample is formed by generating a 64 × 64 image of random white Gaussian noise. This is then low-pass filtered by convolving with a Gaussian of standard deviation of 3 pixels. The resulting image is windowed by multiplication with another Gaussian with standard deviation of 6 pixels. A network with 4096 inputs and 16 outputs is trained on 2000 such input samples. The resulting masks are shown in Figure 9. In Figure 10 we see cross-sections through the major axes of the first, third, and sixth masks of Figure 9. The first cross-section is qualitatively similar to the receptive field shapes of retinal ganglion cells shown in Enroth-Cugell and Robson (1966). The second and third cross-sections appear similar to the receptive field shapes of cortical simple cells shown in Andrews and Pollen (1979). Many of the remaining masks which the network has learned do not correspond to any receptive field shape which has been found in primate visual cortex (see Sanger, in press for further discussion of these issues.)

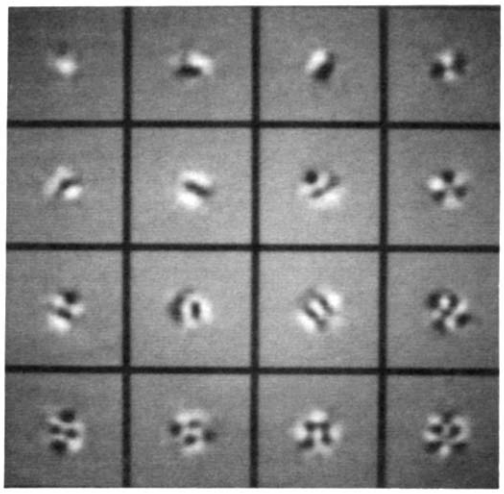

FIGURE 9. First 16 receptive field masks learned by a network with random input. (Order is left-to-right, top-to-bottom.)

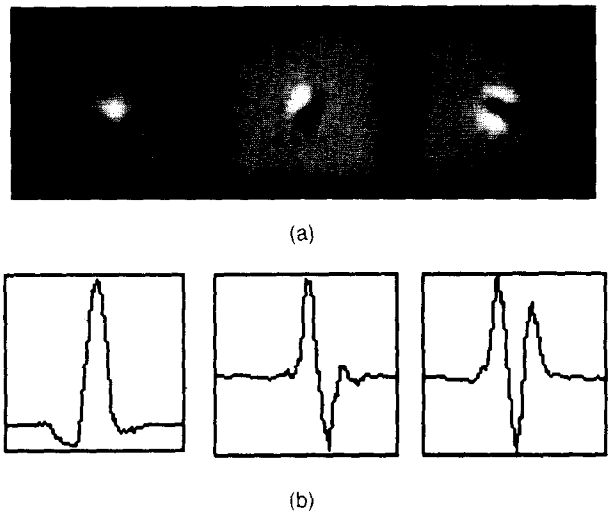

FIGURE 10. (a) First, third, and sixth receptive fields from Figure 9. (b) Cross-sections of (a) through major axes.

These results are not meant to imply that the visual system develops through an adaptation mechanism similar to the Generalized Hebbian Algorithm; we have no evidence for such a claim. Rather, we claim only that our results imply that there exist simple algorithms through which the observed receptive field shapes can develop.

Since the Generalized Hebbian Algorithm finds the eigenvectors of the input distribution, the masks which it learns must be the eigenvectors of the data we presented. Since these masks may be similar to the actual observed receptive fields, it follows that the receptive fields may actually be performing an eigenvector decomposition of the input and therefore have the optimality properties of the KLT (if we assume that visual input can be modeled as bandpass filtered white noise). Note that the masks of Figure 3 and Figure 9 are similar, which indicates that the statistics of the picture in Figure 2 and the random input used to train the network of Figure 9 are related.

This interpretation of the role of the visual cortex unifies several other disparate views of its computations. Some authors believe that the visual cortex recognizes simple features in the environment which represent significant parts of the image such as edges or blobs (e.g., Hubel & Wiesel, 1962; Marr, 1982). Other authors consider the early stages of the visual system to be performing a localized spatial-frequency decomposition of the image, or a sort of "local Fourier transform" (Pollen & Ronner, 1983; Shapley & Lennie, 1985, for review). However, these two seemingly different viewpoints are both equivalent to the observation that simple cells represent the eigenvectors of the input distribution. It is well known that the eigenvectors of any stationary distribution are given by complex exponentials, which is the same basis as that of the Fourier transform (see Kazakos, 1983 and Yuille et al., 1988 for a detailed discussion). Therefore, the Generalized Hebbian Algorithm will learn a set of filters which can perform a spatial-frequency analysis of the input image. Also, if the input image has statistically significant features such as edges, maximization of variance will produce "edge-detectors." By comparing Figure 9 and Figure 8a we see that spatial-frequency detectors and edge detectors can have very similar shapes.

The significance of the filters learned by the Generalized Hebbian Algorithm is thus greater than either spatial frequency analysis or feature detection. Both types of decomposition of the image occur simultaneously due to the maximization of output variance. The algorithm chooses its representation of the input based upon the true statistics of the input. For real scenes, these statistics include features as well as spatial frequency components.

The theoretical mechanism for receptive field development presented here differs from the mechanism proposed by Linsker (1986). Here, all the cells develop in a single layer, whereas Linsker's network requires multiple layers. Each cell in a given layer finds the principal component of the autocorrelation of the previous layer. All the cells of any given layer have similar receptive field shapes. At the seventh layer, orientation-tuned cells develop which have shapes like those shown in Figure 9. Kammen and Yuille (1988) have explained the evolution of oriented cells in Linsker's network in terms of symmetry-breaking caused by unstable critical points in an energy function.

Barrow (1987) has proposed a model of receptive field development which is qualitatively very similar to that presented here. His input is bandpass-filtered white noise, and he hypothesizes that the bandpass operation occurs in retina and lateral geniculate. He uses a Hebbian learning rule and maintains the row norms at one. To avoid having all the outputs converge to the same set of weights, Barrow uses a winner-take-all strategy in which only the strongest output has its weights updated. We suspect that this form of competitive learning may be formally equivalent to the Generalized Hebbian Algorithm. Barrow's algorithm does not order the outputs by decreasing variance, but it is possible that the actual eigenvectors are discovered, rather than a linear combination of them. It is not clear whether or not the outputs of his network are orthogonal, but his results seem very similar to Figures 9 and 10.

Recently, Yuille et al. (1988) have proposed a model for cortical cell development based on a Hebbian learning rule and lateral interactions between pairs of cells. Their algorithm learns quadrature-phase pairs of cells, as are found in primate visual cortex (Pollen & Ronner, 1981). Their equation 24 is similar to the Generalized Hebbian Algorithm, and may be formally equivalent.

CONCLUSION

In this paper we have presented a simple network learning algorithm based on a Hebbian learning rule. Analysis of the algorithm leads to two important new results. First, we proved that the algorithm converges to a network which satisfies an optimality principle based on maximizing the ability to recover the input data. And second, we showed that the algorithm can be implemented using only local network operations.

The algorithm generalizes the Oja (1982) network algorithm to the multiple-output case, and verifies conjectures of Linsker (1988) and Baldi and Hornik (1988) about the existence of such an algorithm. Although several similar algorithms have been proposed in the field of signal processing (Brockett, 1989; Oja & Karhunen, 1985; Owsley, 1978; Oja, 1983, among others), we have shown that there is a fundamental relationship between these algorithms and well-known Hebbian learning rules for neural networks. We have used the Generalized Hebbian Algorithm to generate a local learning rule, and we have shown that it can be an important and useful method for training unsupervised networks.

A network trained according to the Generalized Hebbian Algorithm computes the Karhunen-Loève transform of the input distribution. The well-known properties of the KLT provide a general framework for analyzing the behavior of such networks. We can then make use of the extensive literature in other fields to help understand the behavior of unsupervised neural networks. The relation to statistical procedures such as Factor Analysis or Principal Components Analysis makes clear the fundamental nature of the theory. The power of the KLT allows us to use a simple single-layer linear network to begin an approach to such problems as image coding, texture segmentation, and feature discovery.

Once the applicability of the KLT to neural nets is understood, we can use it to analyze the behavior of hidden units trained in "encoder" networks (Baldi & Hornik, 1989; Bourlard & Kamp, 1988). For many practical applications, the ability of the Generalized Hebbian Algorithm to find uncorrelated outputs of maximal variance gives it advantages over encoder networks trained with self-supervised back-propagation.

Although we have only considered the simplest case of a single-layer linear network, the techniques developed and the interpretation of the network are useful for understanding the behavior of more complex networks. Much work needs to be done to apply this research to multi-layer networks with nonlinear node functions, but the linear case gives us some clues as to how to proceed (Sanger, 1989b).

REFERENCES

Andrews, B. W., & Pollen, D. A. (1979). Relationship between spatial frequency selectivity and receptive field profile of simple cells. *Journal of Physiology*, **287**, 163–176.

Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, **2**, 53–58.

Ballard, D. H. (1987). Modular learning in neural networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (*AAAI-87*) (Vol. 1, pp. 279–284). Los Altos, CA: Morgan Kaufman.

Barrow, H. G. (1987). Learning receptive fields. In *Proceedings of the IEEE 1st Annual Conference on Neural Networks* (Vol. 4, pp. 115–121). Washington, DC: IEEE Computer Society Press.

Bergen, J. R., & Adelson, E. H. (1988). Early vision and texture perception. *Nature*, **333**, 363–364.

Bienenstock, E. L., Cooper, L. N., & Munro, P. W. (1982). Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, **2**, 32–48.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, **59**, 291–294.

Brailovsky, V. (1983a). On the problem of function approximation by sample set processing for an incompletely determined model. *Annals of the New York Academy of Science*, **410**, 137–147.

Brailovsky, V. (1983b). On the problem of function system selection for function approximation based on the use of a sample set with defects. *Annals of the New York Academy of Science*, **410**, 149–161.

Brailovsky, V. L. (1985). On an incompletely determined model for function approximation by experimental data. *Annals of the New York Academy of Science*, **452**, 316–333.

Brockett, R. W. (1989). *Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems*. Unpublished manuscript, Harvard University, Cambridge, MA.

Carpenter, G. A., & Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, **21**, 77–88.

Cottrell, G. W., Munro, P., Zipser, D. (1987). Learning internal representations from gray-scale images: An example of extensional programming. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society* (pp. 461–473).

Daugman, J. G. (1988). Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **36**(7), 1169–1179.

Enroth-Cugell, C., & Robson, J. B. (1966). The contrast sensitivity of retinal ganglion cells of the cat. *Journal of Physiology*, **187**, 517–552.

Fukunaga, K. (1972). *Introduction of statistical pattern recognition*. New York: Academic Press.

Golub, G. H., Van Loan, C. F. (1983). *Matrix computations*. Oxford: North Oxford Academic Press.

Grossberg, S. (1976). On the development of feature detectors in the visual cortex with applications to learning and reaction-diffusion systems. *Biological Cybernetics*, **21**, 145–159.

Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

Hinton, G. E. (1987). *Connectionist learning procedures.* (Tech. Rep. No. CS-87-115). Carnegie-Mellon University, Pittsburgh, PA.

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, **160**, 106–154.

Kammen, D. M., & Yuille, A. L. (1988). Spontaneous symmetry-breaking energy functions and the emergence of orientation selective cortical cells. *Biological Cybernetics*, **59**, 23–31.

Karhunen, J. (1984a). Adaptive algorithms for estimating eigenvectors of correlation type matrices. In *Proceedings of the 1984 IEEE International Conference on Acoustics, Speech, and Signal Processing.* (pp. 14.6.1–14.6.4). New York: IEEE Press.

Karhunen, J. (1984b). *Recursive estimation of eigenvectors of correlation type matrices for signal processing applications.* Ph.D. thesis, Helsinki University of Technology, Espoo, Finland.

Karhunen, J. (1985). *Simple gradient type algorithms for data-adaptive eigenvector estimation.* (Tech. Rep. No. TKK-F-A584). Finland: Helsinki University of Technology.

Karhunen, J., & Oja, E. (1982). New methods for stochastic approximation of truncated Karhunen-Loève expansions. In *Proceedings of the 6th International Conference on Pattern Recognition* (pp. 550–553). New York: Springer-Verlag.

Kazakos, D. (1983). Optimal constrained representation and filtering of signals. *Signal Processing*, **5**, 347–353.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, **43**, 59–69.

Kohonen, T. (1988). The "neural" phonetic typewriter. *Computer*, **21**, 11–22.

Kreyszig, E. (1988). *Advanced engineering mathematics.* New York: Wiley.

Kushner, H. J. & Clark, D. S. (1978). *Stochastic approximation methods for constrained and unconstrained systems.* New York: Springer-Verlag.

Kuusela, M., & Oja, E. (1982). The averaged learning subspace method for spectral pattern recognition. In *Proceedings of the 6th International Conference on Pattern Recognition* (pp. 134–137). New York: Springer-Verlag.

Lim, J. S. (in press). *Two-dimensional signal and image processing.* Englewood Cliffs, NJ: Prentice Hall.

Linsker, R. (1986). From basic network principles to neural architecture. *Proceedings of the National Academy of Science, USA*, **83**, 7508–7512, 8390–8394, 8779–8783.

Linsker, R. (1988). Self-organization in a perceptual network. *Computer*, **21**, 105–117.

Lippmann, R. P., (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, **4**, 4–22.

Ljung, L. (1977). Analysis of recursive stochastic algorithms. *IEEE Transactions on Automatic Control*, **AC-22**, 551–575.

Marr, D. (1982). *Vision.* San Francisco: W.H. Freeman and Co.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematics and Biology*, **15**, 267–273.

Oja, E. (1983). *Subspace methods of pattern recognition.* Letchworth, Hertforshire, UK: Research Studies Press.

Oja, E., & Karhunen, J. (1980). Recursive construction of Karhunen-Loève expansions for pattern recognition purposes. In *Proceedings of the 5th International Conference on Pattern Recognition* (pp. 1215–1218). New York: Springer-Verlag.

Oja, E., & Karhunen, J. (1985). On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications*, **106**, 69–84.

Owsley, N. L., (1978). Adaptive data orthogonalization. In *Proceedings of the 1978 IEEE International Conference on Acoustics, Speech, and Signal Processing* (pp. 109–112). New York: IEEE Press.

Pollen, D. A., & Ronner, S. F. (1981). Phase relationships between adjacent simple cells in the visual cortex. *Science*, **212**, 1409–1411.

Pollen, D. A., & Ronner, S. F. (1983). Visual cortical neurons as localized spatial frequency filters. *IEEE Transactions on Systems, Man, and Cybernetics*, **13**, 907–916.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (Eds.). (1986a). Learning internal representations by error propagation. In *Parallel distributed processing* (chap. 8, pp. 318–362). Cambridge, MA: MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams R. J. (1986b). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Sanger, T. D. (1988). Optimal unsupervised learning. *Neural Networks*, **1** (Supl. 1), 127.

Sanger, T. D. (in press). Neural networks, principal components analysis, and Gabor filters in low-level vision. *Biological Cybernetics.*

Sanger, T. D. (1989b). An optimality principle for unsupervised learning. In D. Touretzky (Ed.), *Advances in neural information processing systems.* San Mateo, CA: Morgan Kaufmann.

Shapley, R., & Lennie, P. (1985). Spatial frequency analysis in the visual system. *Annual Review of Neuroscience*, **8**, 547–583.

Silverman, R. H., & Noetzel, A. S. (1988). Time-sequential self-organization of hierarchical neural networks. In D. Z. Anderson (Ed.), *Neural information processing systems* (pp. 709–714). New York: American Institute of Physics.

Sontag, E. D. (1988). *Some remarks on the backpropagation algorithm for neural net learning* (Tech. Rep. No. SYCON-88-2). New Jersey: Rutgers Center for Systems and Control.

Sontag, E. D., & Sussmann, H. J. (1988). *Backpropagation can give rise to spurious local minima even for networks without hidden layers* (Tech. Rep. No. SYCON-88-08). New Jersey: Rutgers Center for Systems and Control.

Turner, M. R. (1986). Texture discrimination by Gabor functions. *Biological Cybernetics*, **55**, 71–82.

Voorhees, H., & Poggio T. (1988). Computing texture boundaries from images. *Nature*, **333**, 364–367.

Voorhees, H. (1987). *Finding texture boundaries in images* (Tech. Rep. No. 968). MIT AI Lab, Cambridge, MA.

Watanabe, S. (1965). Karhunen-Loève expansion and factor analysis: Theoretical remarks and applications. *Transactions of the 4th Prague Conference on Information Theory* (pp. 635–660). Prague: Publishing House of the Czechoslovak Academy of Sciences.

Yuille, A. L., Kammen, D. M., & Cohen, D. (in press). Quadrature and the development of orientation selective cortical cells by Hebb rules. *Biological Cybernetics.*